Second Half Review

Siva Balakrishnan
Data Mining: 36-462/36-662

April 25th, 2019

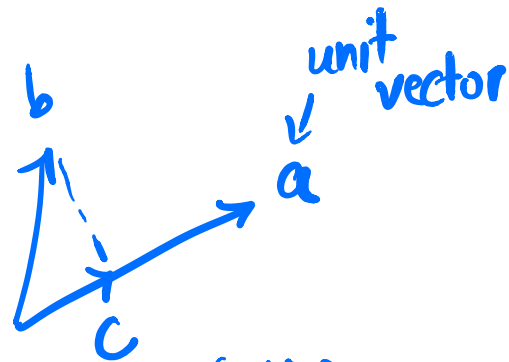
Plan for the rest of the semester

- Exam review today – please ask questions!
- HW due on Wednesday at midnight
- Remember the project deadlines (i.e. next Tuesday predictions are due, and next Friday write-up is due)
- Proposals for next week:
  1. Association rules (guest lecture by CPM)
  2. Recommendation systems, bandits, online learning
  3. Reinforcement learning
  4. Dealing with text data (featurization, topic models)

Linear Algebra Review

- Vectors:

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{bmatrix}.$$



- The length of a vector:

$$\|v\|_2 = \sqrt{v_1^2 + \ldots + v_d^2}.$$

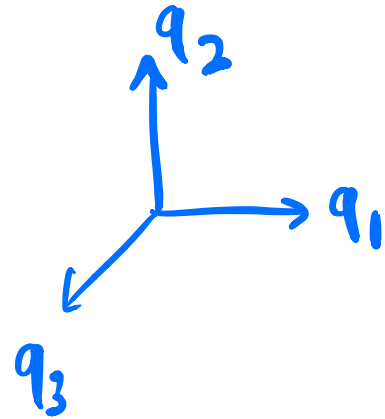
- The projection of a vector b onto a *unit* vector a:

$$\text{proj}_a(b) = (a^T b)a.$$

Orthonormal Matrices

- Matrices $Q \in \mathbb{R}^{d \times d}$:

$$Q = \begin{bmatrix} \uparrow & \uparrow & \cdots & \uparrow \\ q_1 & q_2 & \cdots & q_d \\ \downarrow & \downarrow & \cdots & \downarrow \end{bmatrix},$$



which satisfy:

$$q_i^T q_j = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{otherwise.} \end{cases}$$

$q_i^T q_j = 0$. $i \neq j$

- Orthonormal matrices satisfy:

$$Q^T Q = I,$$
$$Q Q^T = I,$$
$$Q^{-1} = Q^T.$$

Matrix Decompositions

- Every real, symmetric matrix M can be *diagonalized*, i.e. we can write:

$$M = U \times D \times U^T,$$

U is an orthonormal matrix.

for a *diagonal* matrix D, and an *orthnormal* matrix U.

- The columns of U are called *eigenvectors*, and each column of U has an associated diagonal entry in the matrix D are that is its associated *eigenvalue*.
- We will usually arrange things so that $|D_{11}| \geq |D_{22}| \geq \ldots$. Positive semi-definite matrices are ones for which every eigenvalue is ≥ 0.
- The eigendecomposition has many uses. Given the eigendecomposition you can easily invert the matrix, raise it to some power, compute the matrix exponential and so on.
- We will also see that it will give us crucial insight into important matrices.

Some Eigenvector Properties

- Suppose M is real, symmetric and we write:

$$M = U \times D \times U^T,$$

for a *diagonal* matrix D, and an *orthnormal* matrix U.

- We can see that:

$$Mu_i = d_{ii}u_i,$$

for any column u_i of U.

- The top eigenvector maximizes the quadratic form $v^T M v$, i.e.:

$$u_1 = \arg \max_{\|v\|_2=1} v^T M v,$$

and the bottom eigenvector minimizes it, i.e.:

$$u_d = \arg \min_{\|v\|_2=1} v^T M v.$$

More Matrix Decompositions

- Every real matrix (not necessarily symmetric or even square) M can be written in terms of its *Singular Value Decomposition*:

$$M = U \times \Sigma \times V^T,$$

U & V are orthonormal

Σ diagonal.

for a *diagonal* matrix Σ with all positive entries, and two *orthnormal* matrices U, V.

- In particular, we can see that:

$$MM^T = U \times \Sigma^2 \times U^T$$
$$M^T M = V \times \Sigma^2 \times V^T.$$

So U and V are just the eigenvectors of MM^T and $M^T M$ (which are both symmetric matrices).

Recap: Principal Components Analysis (PCA)

want an interesting $\mathbb{R}^{n\times 2}$ matrix.

- In unsupervised learning, we are just given a (big) data matrix $X \in \mathbb{R}^{n\times p}$.
- A basic question is: can we (meaningfully) reduce the dimension of the data either so we can visualize it, cluster it, or even do better supervised learning with it.
- PCA answers this questions by finding "interesting directions" and projecting the data on to those directions.
- It is the most widely used exploratory data analysis tool. It is extremely useful!

Recap: What are principal components?

- The linear algebraic answers:
  1. They are just eigenvectors of the covariance matrix $\widehat{\Sigma} = X^T X/n$.
  2. Equivalently, they are the right singular vectors of the matrix X.

always mean center X.

$\widehat{\Sigma} \in \mathbb{R}^{p \times p}$.

$\widehat{\Sigma} = V D V^T$ → cols of V are called PCs.

$X = U \tilde{D} V^T$

X – subtract column mean.

$$\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T.$$

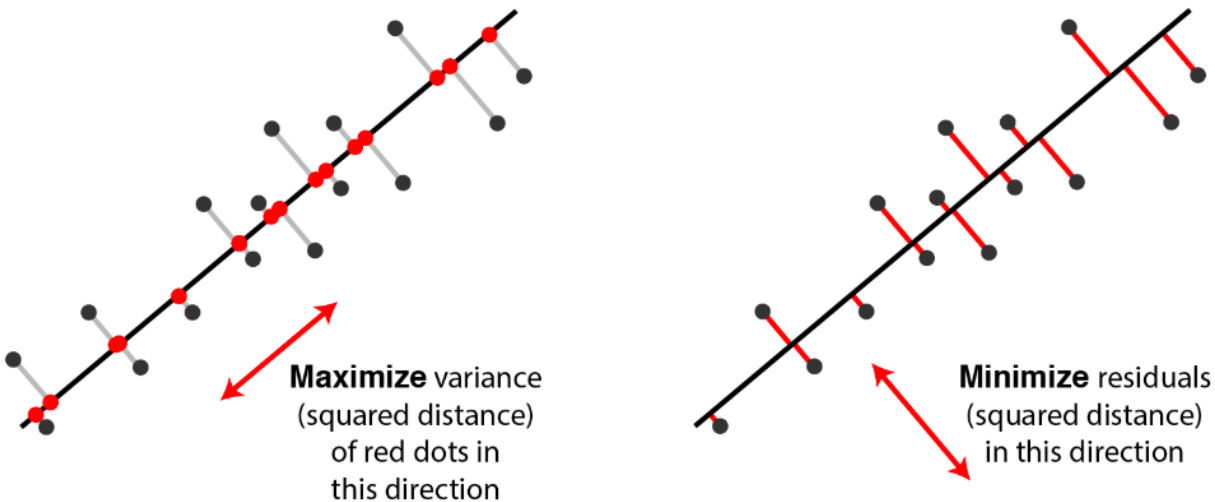
Recap: What are principal components?

- The statistics/data-based answers:
  1. They are the directions of maximum variance. For instance, the first principal component:

$$v_1 = \arg\max_{\|v\|_2 = 1} v^T \hat{\Sigma} v := \frac{1}{n} \sum_{i=1}^{n} (x_i^T v)^2.$$

variance in the dir. v.

  2. They are subspaces which are closest on average to the data.



Amount/Proportion of Variance Explained

- Suppose we write:

$$\hat{\Sigma} = V D V^T$$

then the:

1. Total variance in the data is given by:

$$T = \sum_{i=1}^{d} d_{ii}.$$

2. Variance explained by i-th principal component is given by:

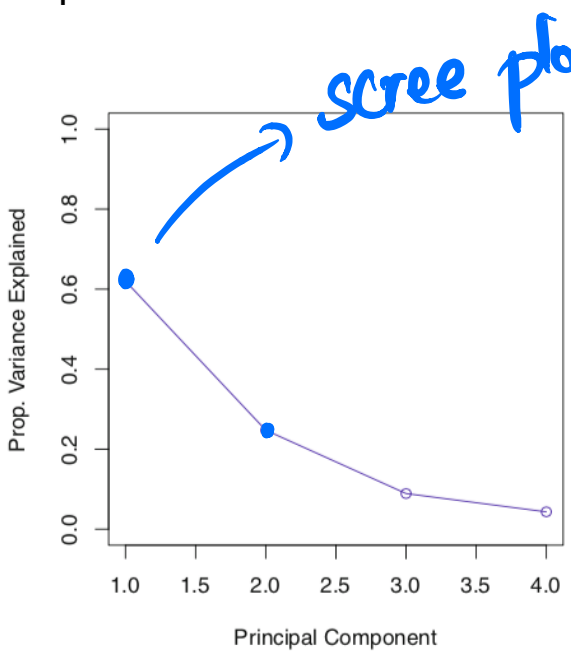
$$d_{ii}.$$

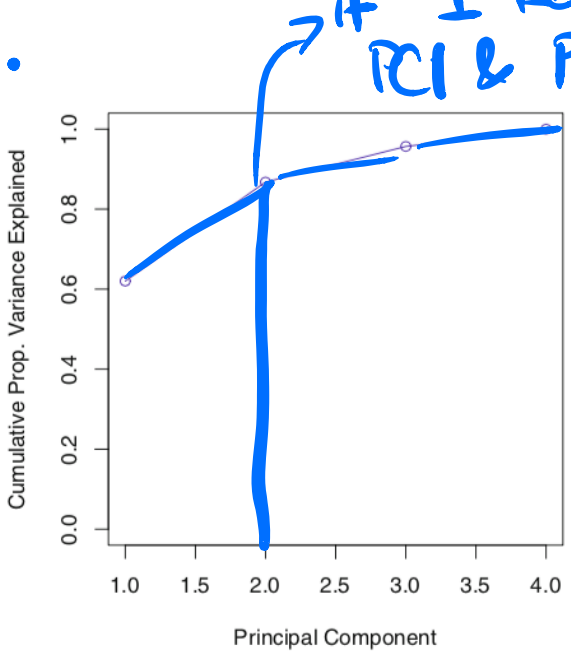
3. Cumulative proportion of variance explained:

$$= \frac{\sum_{i=1}^{k} d_{ii}}{\sum_{i=1}^{k} d_{ii}}$$

Recap: Amount/Proportion of Variance Explained

- Leads to two visualizations of the value of added principal components:



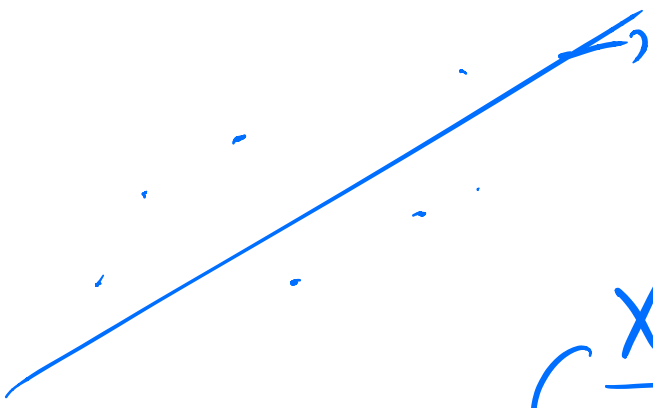


Recap: Dimension Reduction/Visualization

- Suppose we want to visualize our data (or reduce its dimensionality) in k dimensions.
- We simply compute the projection of our data onto the k PCs and plot it:

$$X = U \tilde{D} V^T.$$

These are called PC scores. cols. of $U\tilde{D}$ are the visualization.



$X \in \mathbb{R}^{n \times p}$.

$v_1 \in \mathbb{R}^p$.

$Xv_1 \in \mathbb{R}^n$.

↳ "new feature".

Recap: k-means

- One way to cluster data is sometimes called *objective based clustering*.
  1. We write down an intuitive way to measure the quality of a given clustering (our objective).
  2. We then try to derive an algorithm to find a good clustering (as measured by our objective).
- k-means is such a method.

Recap: k-means

- Given a set of points $X_1, \ldots, X_n$ and dissimilarites $d(X_i, X_j) = \|X_i - X_j\|_2^2$.
- Denote a clustering by a function C which maps data points to $\{1, \ldots, K\}$.
- Three equivalent objectives:
  1. **Within-cluster scatter:**
  2. **Within-cluster variation:**
  3. **Modified within-cluster variation:**
- A good clustering has low value of these objectives.

Recap: Minimizing the objective – Lloyd's algorithm

- It is hard to minimize the k-means objective over all possible clusterings of the data. A popular heuristic is to use alternating minimization.
- We want to minimize

$$\min_{C} \min_{c_1 \dots c_K} \sum_{k=1}^{K} \sum_{C(i)=k} d(X_i, c_k).$$

$$\sum_{k=1}^{K} \sum_{C(i)=k} \|X_i - c_k\|_2^2,$$

over both clusterings C and $c_1, \dots c_K \in \mathbb{R}^p$.

- Minimize it just over C (keep $c_1, \dots, c_K$ fixed)?

$$C(i) \leftarrow \arg\min_{\{c_1 \dots c_K\}} d(X_i, c_k)$$

Minimize it just over $c_1, \dots, c_K$ (keep C fixed)?

$$c_i = \frac{1}{n_k} \sum_{i \in C_k} X_i.$$

Recap: Lloyd's algorithm

We start with an initial guess for $c_1, \ldots c_K$ (e.g., pick K points at random over the range of $X_1, \ldots X_n$), then repeat:

1. Minimize over C: for each $i = 1, \ldots n$, find the cluster center c_k closest to X_i, and let $C(i) = k$
2. Minimize over $c_1, \ldots c_K$: for each $k = 1, \ldots K$, let $c_k = \bar{X}_k$, the average of points in group k

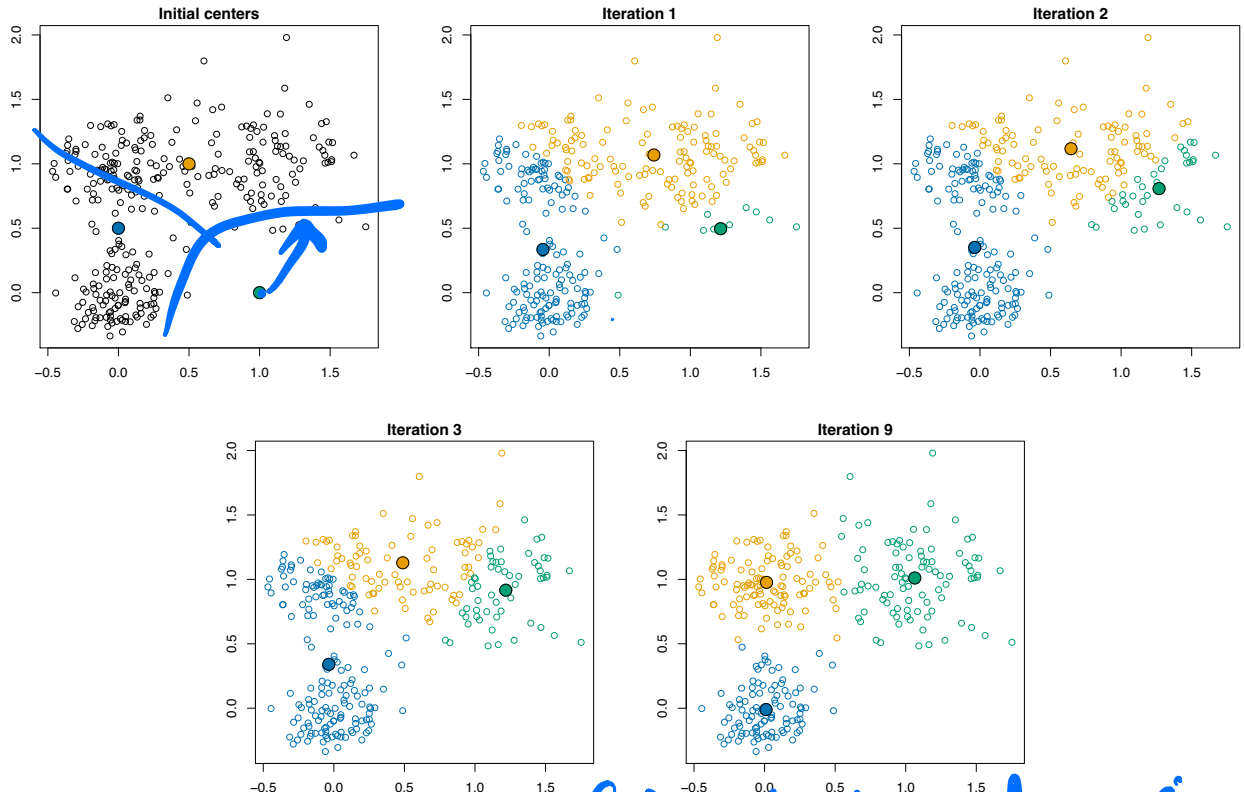
Stop when cluster assignments/within-cluster variation do not change.

In words:

1. Cluster (label) each point based the closest center
2. Replace each center by the average of points in its cluster

Recap: K-means example

Here $X_i \in \mathbb{R}^2$, $n = 300$, and $K = 3$



Nice things: fast, $W(C)$ will keep decreasing

Some drawbacks: Randomized, won't find best clustering

Recap: K-medoids

- Just like K-means except we want the centers $c_1, \ldots, c_K$ to be actual data points.
- Initial guess for centers $c_1, \ldots c_K$ (e.g., randomly select K of the points $X_1, \ldots X_n$), then repeat:

exactly as in K-means

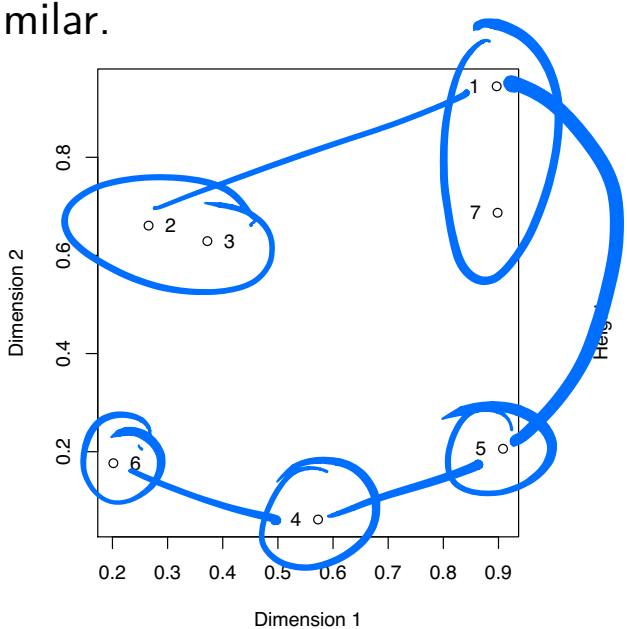
1. Minimize over C: for each $i = 1, \ldots n$, find the cluster center c_k closest to X_i, and let $C(i) = k$
2. Minimize over $c_1, \ldots c_K$: for each $k = 1, \ldots K$, let $c_k = X_k^*$, the medoid of points in cluster k, i.e., the point X_i in cluster k that minimizes $\sum_{C(j)=k} \|X_j - X_i\|_2^2$

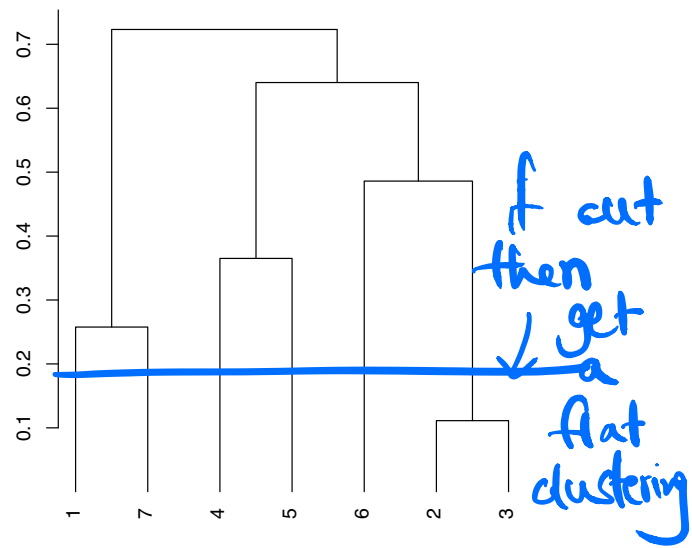
Stop when within-cluster variation doesn't change

- Advantages over K-means: Representative of cluster is a data point.
- Disadvantages relative to K-means: → much more expensive computationally.

Recap: Hierarchical Clustering

- Want to produce a sequence of *nested* clusters. No need to specify K anymore.
- Two broad strategies: agglomerative and divisive.
- Represent hierarchical clusters by a dendrogram: cut horizontally to get clusters, and heights tell us about (dis)similarities, groups that merge near the bottom are quite similar.





Recap: Linkage

- To specify an agglomerative hierarchical clustering algorithm we *only specify one thing*: the linkage rule. This is just a way to assign a distance to two *groups* of points (usually derived from a distance between individual points).
- The most canonical ways to do this:
  1. Single Linkage:

$$d_{\text{single}}(G, H) = \min_{i \in G,\, j \in H} d_{ij}$$

  2. Complete Linkage:

$$d_{\text{complete}}(G, H) = \max_{i \in G,\, j \in H} d_{ij}$$

  3. Average Linkage:

$$d_{\text{average}}(G, H) = \frac{1}{n_G n_H} \sum_{i \in G,\, j \in H} d_{ij}$$

Recap: Cut Interpretations

- Suppose we cut a (single/complete/average)-linkage dendrogram at some height h to get clusters. Can we say anything nice about the clusters?
- **Single Linkage:** if $x \in C_1$ $y \in C_2$. $d(x,y) > h$
- **Complete Linkage:** if $x \in C_1, y \in C_1$ $d(x,y) \leq h$.
- **Average Linkage:** Nothing particularly interesting to say here.

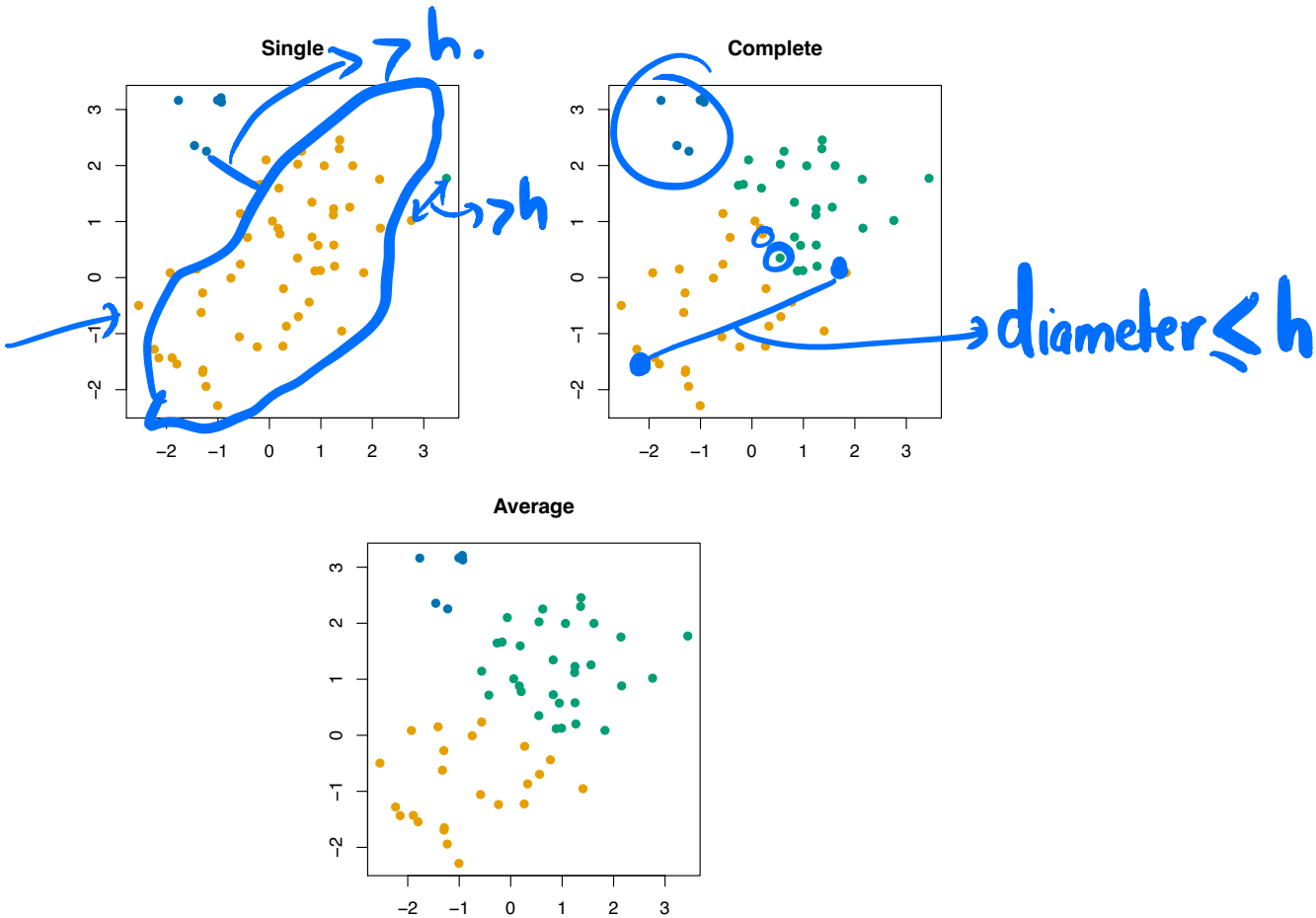
Shortcomings of single, complete linkage

Single and complete linkage can have some practical problems:

- Single linkage suffers from chaining. In order to merge two groups, only need one pair of points to be close, irrespective of all others. Therefore clusters can be too spread out, and not compact enough
- Complete linkage avoids chaining, but suffers from crowding. Because its score is based on the worst-case dissimilarity between pairs, a point can be closer to points in other clusters than to points in its own cluster. Clusters are compact, but not far enough apart

Average linkage tries to strike a balance. It uses average pairwise dissimilarity, so clusters tend to be relatively compact and relatively far apart

Example of chaining and crowding



Linkages summary

Linkage	No inversions?	Unchanged with monotone transformation?	Cut interpretation?	Notes
Single	✓	✓	✓	chaining
Complete	✓	✓	✓	crowding
Average	✓	×	×	
Centroid	×	×	×	simple

Note: this doesn't tell us what "best linkage" is.

Remember that choosing a linkage can be very situation dependent.

Recap: Mixture Models Motivation

- We wanted to fix two significant problems with K-means clustering:
  - It is a "hard" clustering method, i.e. each point gets assigned to a single cluster and so deals badly with overlapping clusters.
  - It can also do poorly in cases where the clusters have non-spherical shapes.

- **Bonus:** Perhaps incorporate a bit more "statistical modeling" into clustering.

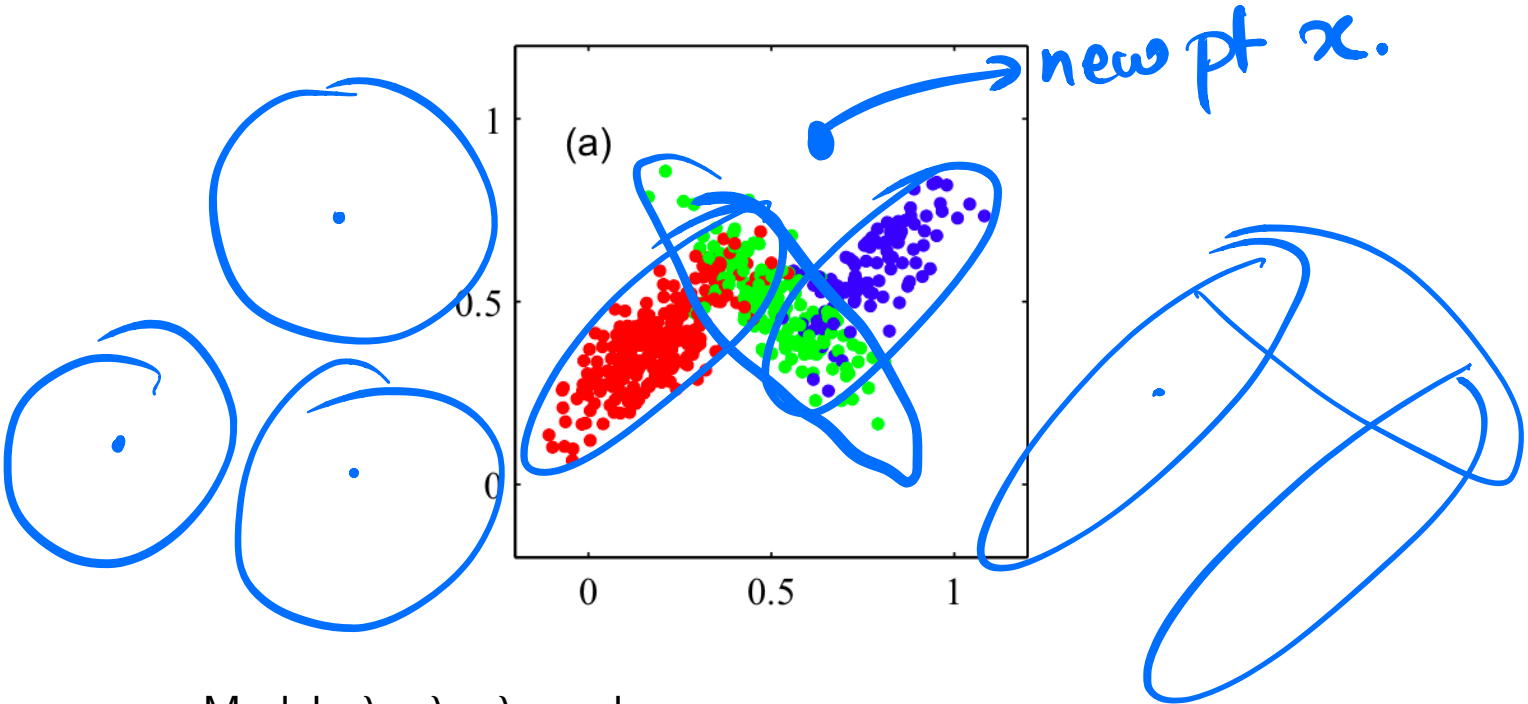
Recap: Mixture Models

- Want to roughly imagine the case, where each cluster has a different distribution.
- The generative model we are imagining is:
  - We first choose a cluster by drawing $Z \sim \{1, \ldots, K\}$.
  - We then draw a sample from the distribution corresponding to cluster Z.

  However, we are not shown the Z values (the cluster labels).
- This is called a *mixture model*:

$$f(x) = \sum_{k=1}^{K} \mathbb{P}(Z = k) p(x | Z = k) = \sum_{k=1}^{K} \lambda_k f_Z(x).$$

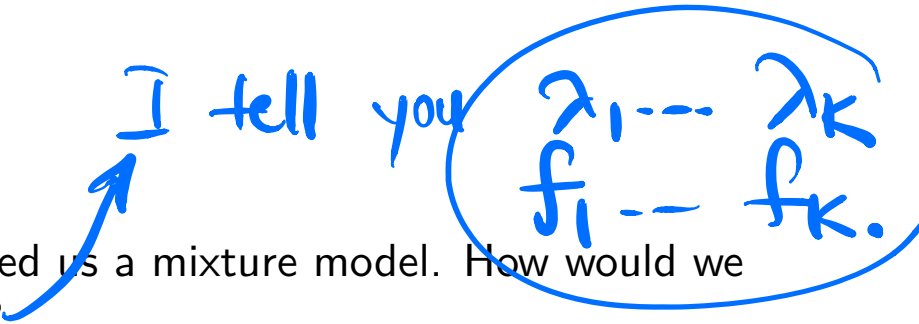
Recap: Gaussian Mixture Models



- Model: $\lambda_1, \lambda_2, \lambda_3$ and

$$f_{\text{red}}(x) = N(\mu_{\text{red}}, \Sigma_{\text{red}}),$$
$$f_{\text{blue}}(x) = N(\mu_{\text{blue}}, \Sigma_{\text{blue}}),$$
$$f_{\text{green}}(x) = N(\mu_{\text{green}}, \Sigma_{\text{green}}),$$

Recap: Clustering with a Mixture Model



- Suppose someone handed us a mixture model. How would we "soft" cluster our data?
- For a point x we would compute for $i \in \{1, \ldots, K\}$:

$$P(Z = i | X = x) = \frac{f_i \lambda_i}{\sum_{j=1}^{K} f_j \lambda_j}.$$

Main question: given data how do we estimate the mixture parameters?

Recap: Estimating a Mixture Model – Expectation-Maximization

- EM is a general method for (approximately) maximizing the (marginal) likelihood when you have missing data. We won't get too much into the details but describe the EM algorithm for GMMs directly.

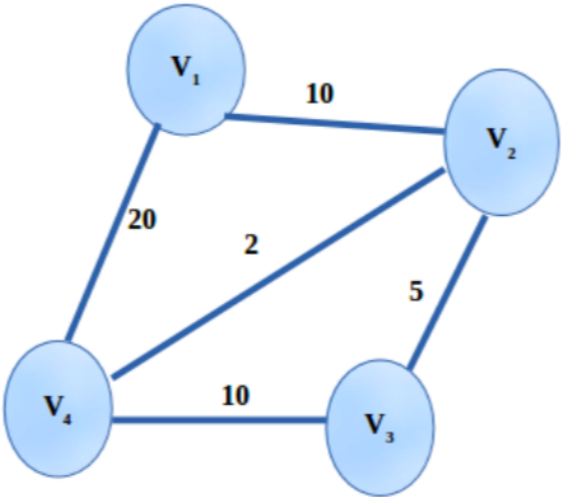
- Roughly, we want to first "guess" the latent variables Z_i and then if we knew those we could just maximize the (usual/complete) likelihood.
- It resembles k-means. Except instead of assigning each point to a single cluster we "softly" assign them so they contribute fractionally to each cluster.

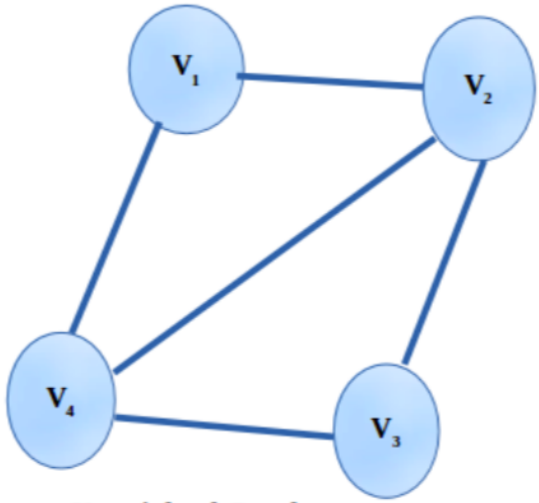
Recap: Graphs

It is often convenient and useful to think about data in terms of graphs.

- **(Unweighted) Graphs:** Just vertices and edges. Equivalent to every edge having weight 1.
- **Weighted Graphs:** Each edge, say between vertices i and j, has weight w_{ij}.

For us, graphs will usually be **undirected** (i.e. the edges do not have an orientation), and weights will usually be positive.

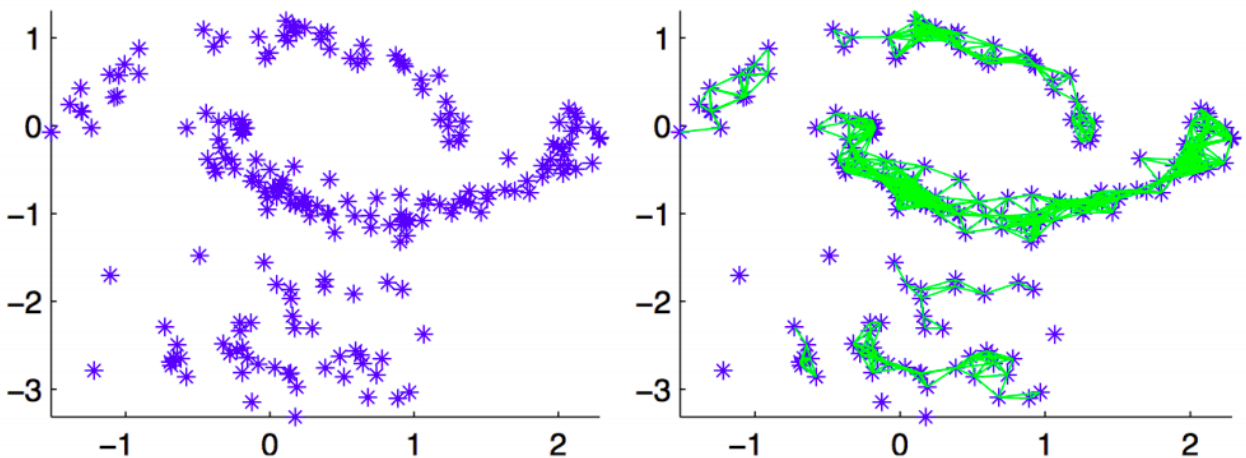




Recap: From Data to Graphs

We are given our usual collection of data points $\{X_1, \ldots, X_n\}$.
How do we build a graph from these?
Roughly:

1. **Nodes:** These are the data points.
2. **Edges/Weights:** We want to connect points that are similar. Weights will measure "similarity".



Recap: From Data to Graphs

Three canonical ways:

- ϵ-**neighborhood graph:**
- k-**NN graph:**
- **Weighted similarity graph:**

Recap: Clustering Graphs

First Attempt: Find best cut, i.e.:

$$\min_{A,B} \text{cut}(A, B) = \min_{A,B} \sum_{i \in A, j \in B} w_{ij}.$$

Does poorly in practice.

Second Attempt: Find best balanced cut:

$$\min_{A,B \text{ of equal size}} \text{cut}(A, B).$$

Does very well in practice. However, hard to compute. Spectral clustering is a fast, approximate way to find a balanced cut.

Recap: The Graph Laplacian

The matrix:

$$L = D - W,$$

is called the *Graph Laplacian*. It is a symmetric, real-valued matrix, so it has an eigendecomposition. We have already seen that for any vector v:

$$v^T L v = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij}(v(i) - v(j))^2 \geq 0,$$

so all its eigenvalues are positive.

Intuition: Bottom few eigenvectors are very smooth (as close to constant as they can be) and are very useful for partitioning the graph.

Recap: Cuts and Eigenvectors

If we want to find the minimum balanced cut we can instead solve the following problem:

$$\arg\min_v v^T(D - W)v$$

- Entries of v are all $\{+1, -1\}$ (so it is a cut vector).
- Entries of v sum to 0 (so it is balanced).

Spectral clustering into two clusters is a *relaxation* of this. We solve:

$$\arg\min_v v^T(D - W)v$$

- Entries of v sum to 0 (so it is balanced),

and then threshold the entries of v to find our clusters.

Recap: Basic Spectral Clustering Algorithm

If we want to cluster our data into two clusters we will follow these steps:

- Build a (weighted) graph on the data points (in one of three ways).
- Construct the graph Laplacian matrix, i.e. compute the matrix $D - W$.
- Find its second-smallest eigenvector v_2.
- Threshold its entries to find the clusters, i.e. take $A = \{i : v_2(i) > 0\}$, and $B = \{i : v_2(i) \leq 0\}$.

The second smallest eigenvector of the Laplacian has its own name (Fiedler vector).

Recap: Algorithm for clustering into k-clusters

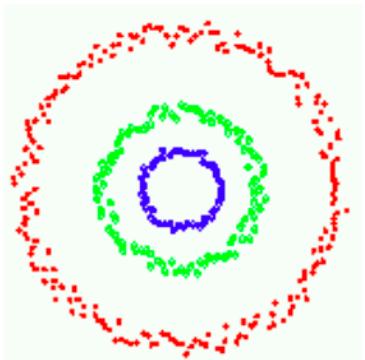
If we want to cluster our data into k-clusters we will follow these steps:

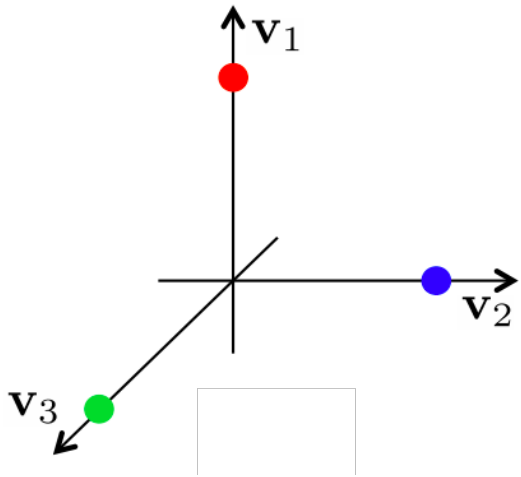
- Build a (weighted) graph on the data points (in one of three ways).
- Construct the graph Laplacian matrix, i.e. compute the matrix $D - W$.
- Find its smallest k eigenvectors $\{v_1, v_2, \ldots, v_k\}$, put them in a matrix $V \in \mathbb{R}^{n \times k}$.
- Interpret the rows of V as our data points. Run k-means on this data to find k-clusters.

Key Point: Spectral Embedding (i.e. mapping into coordinates defined by the eigenvectors of the Laplacian) tends to separate graph clusters.

Recap: The Spectral Embedding

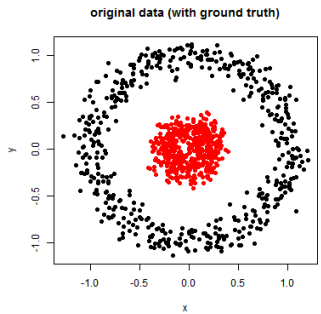
Just some intuition through pictures:

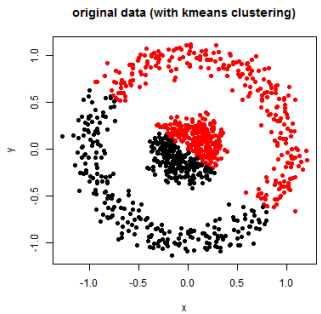


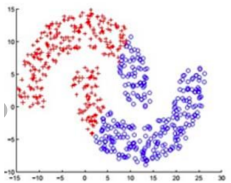


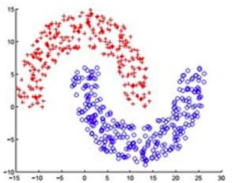
Key point: k-means on the embedding performs much better than k-means on the original data.

Recap: Spectral clustering v/s k-means









- If clusters are (well-separated) blobs: k-means will do well but so will spectral clustering.
- If clusters are dense but have strange shapes then spectral clustering will typically do much better.

Recap: Spectral Clustering in Practice

- People often use the *normalized Laplacian* instead of the graph Laplacian.
- The normalized Laplacian divides each entry of the Laplacian by the square root of the degrees of the two corresponding nodes, i.e.:

$$L_{\text{normalized}} = D^{-1/2} L D^{-1/2}.$$

So,

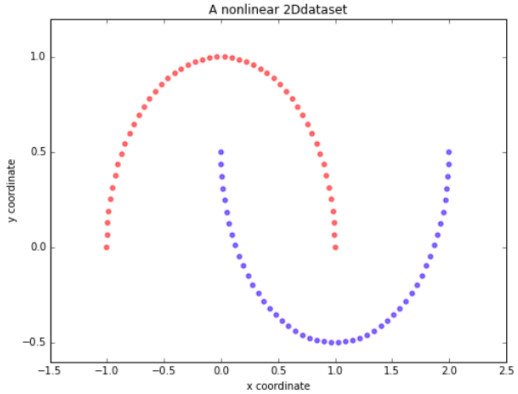
$$L_{\text{normalized}}(i, j) = \frac{L(i, j)}{\sqrt{d_i d_j}}.$$

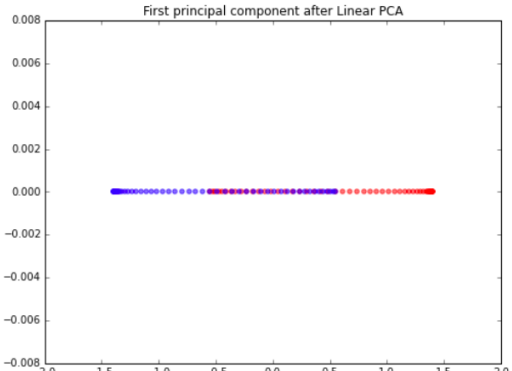
The normalized Laplacian has many of the same properties as the graph Laplacian but tends to give better clusters in practice.

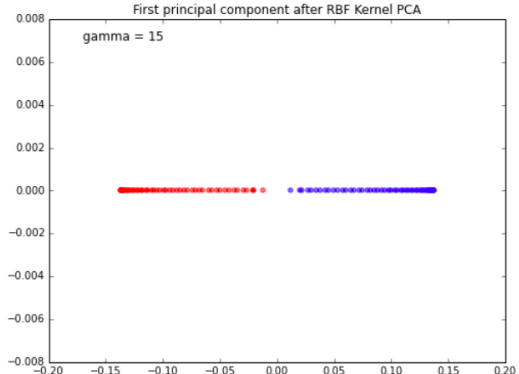
- The rest of the algorithm is identical (compute the bottom k eigenvectors and run k-means on them).

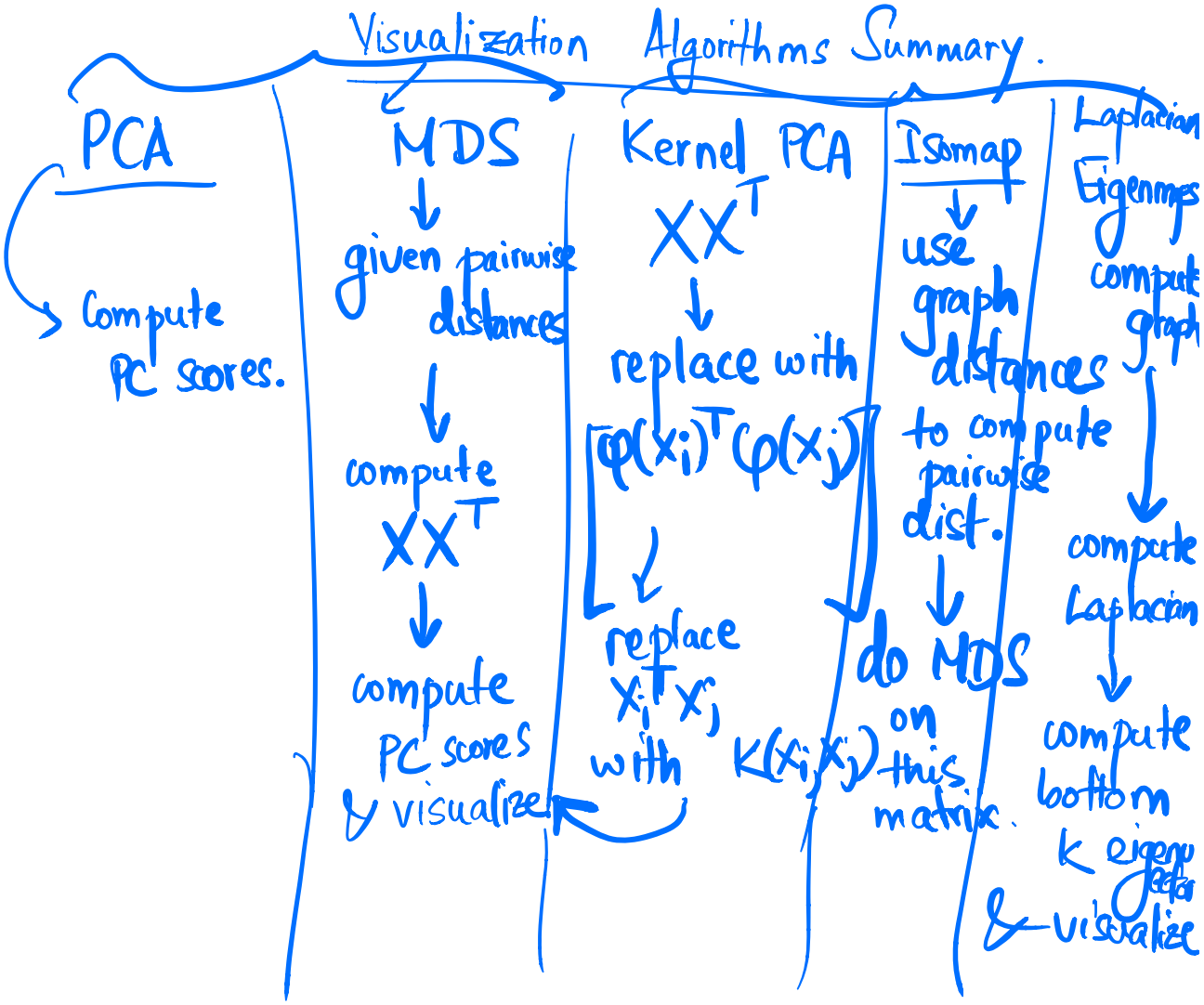
Non-linear Dimension Reduction

- Just like non-linear classifiers can be flexible and useful, we often want to construct non-linear mappings of our data to a lower dimensional space for visualization.
- PCA is a linear dimension reduction method.



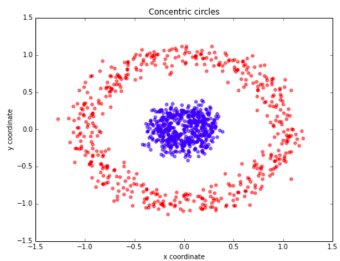


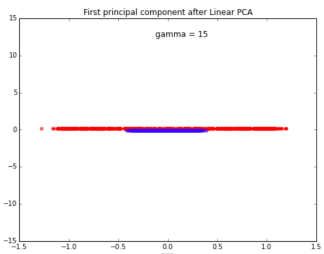


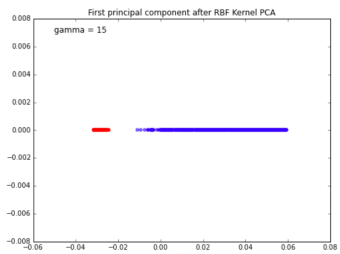

Visualization Algorithms Summary.
PCA
Compute PC scores.
MDS
given pairwise distances
compute XX^T
compute PC scores & visualize
Kernel PCA
XX^T
replace with $\varphi(x_i)^T\varphi(x_j)$
replace $x_i^T x_j$ with $K(x_i, x_j)$
Isomap
use graph distances to compute pairwise dist.
do MDS on this matrix.
Laplacian Eigenmaps
compute graph
compute Laplacian
compute bottom k eigenvectors & visualize

Kernel PCA

- Observe that to compute the PCA projection we do not need X, just the inner-product matrix XX^T suffices.
- Can use the kernel-trick: replace $x_i^T x_j$ by $K(x_i, x_j)$ for some kernel. This gives us the kernel gram matrix K, and we can compute its top few eigenvectors to get the PC scores and visualize our data.



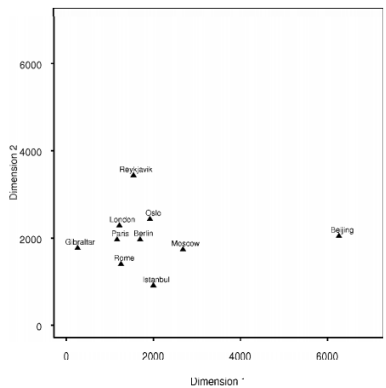




From pairwise distances to visualization

- Given just the matrix of pairwise distances $\Delta_{ij} = \|X_i - X_j\|_2$ we can visualize our data. This is called classical *multi-dimensional scaling*.

	London	Berlin	Oslo	Moscow	Paris	Rome	Beijing	Istanbul	Gibraltar	Reykjavik
London	–									
Berlin	570	–								
Oslo	710	520	–							
Moscow	1550	1000	1020	–						
Paris	210	540	830	1540	–					
Rome	890	730	1240	1470	680	–				
Beijing	5050	4570	4360	3600	5100	5050	–			
Istanbul	1550	1080	1520	1090	1040	850	4380	–		
Gibraltar	1090	1450	1790	2410	960	1030	6010	1870	–	
Reykjavik	1170	1480	1080	2060	1380	2040	4900	2560	2050	–

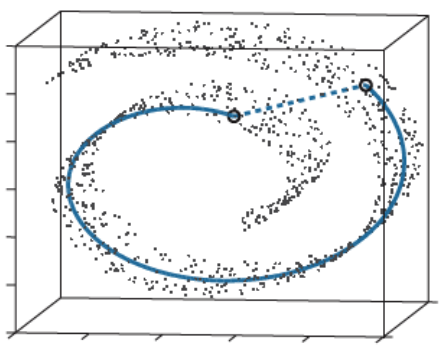


- Key idea: use some transformations to go from $\Delta \to XX^T$ and then compute the PC scores.

Recap: From non-Euclidean pairwise distances to visualization

There is a class of methods which construct a fancier metric Δ_{ij} between high-dimensional points $x_1, \ldots x_n \in \mathbb{R}^p$, and then they feed these Δ_{ij} through multidimensional scaling to get a low-dimensional representation $z_1, \ldots z_n \in \mathbb{R}^k$.

In this case, we don't just get principal component scores, and our low-dimensional representation can end up being a nonlinear function of the data



Recap: Isomap Algorithm

- Build graph.
- Compute shortest path distances.
- Perform MDS on shortest path distances to obtain visualization.

Recap: Laplacian Eigenmaps

We can derive another non-linear visualization method by trying to optimize the locations of the points so as to not distort small distances. This leads to Laplacian Eigenmaps.

- Build graph.
- Find Laplacian, compute bottom k eigenvectors $V \in \mathbb{R}^{n \times k}$.
- Visualize using V.

The same reason why V helps us in clustering applies here, it helps us in visualization by non-linearly preserving clusters.

Recap: Dealing with Data

Lots of considerations in analyzing data:

1. Create train, validation, test folds
2. Useful to understand your data (make plots, make conditional plots, etc.)
3. Understand the task you want to accomplish with data, and constraints (test time budget, train time budget)
4. Fix/assess outliers and missing-data
5. Think carefully about featurization

Recap: Possible Actions

Once you fit a model there are several possible next steps:

- Get more data
- Make more/better features
- Use a more flexible model
- Use a more regularized/less flexible model

Need to use train and validation errors, and possibly a more detailed error analysis (look at individual points) to decide what the next steps are.

Recap: Model Tuning

1. Train-validation-test splits
2. Featurization
3. Quick and dirty (fit a couple of models). Understand base rates (fit naive predictors), understand what error metric you should be using, know that you can trade-off things like precision and recall in many predictors.
4. Diagnose bias/variance problems (use sample-size curves, model-complexity curves, regularization curves, compare different models)
5. Fix bias/variance problems (different set of fixes in each case). Iterate 2,4,5. Think carefully about how to not get bogged down by the tyranny of tuning parameters (use smaller data sets, be parsimonious in choices to try out).
6. Error analysis (diagnose points on which you are predicting poorly, are they outliers? Can you design useful features for them?).
7. Maybe you need more training data but this should be last resort.

Recap: Neural Network Motivation

- One of the most challenging aspects of designing good classifiers is coming up with good/useful features, i.e. transformations/representations of the input that make learning a good classifier easy.
- Neural networks, roughly, try to *learn* useful transformations of the data that are useful. Surprisingly, this often works (at least when you have enough data, enough compute, and know what you are doing).

Recap: Neural Net Basics

Suppose we want to use a representation:

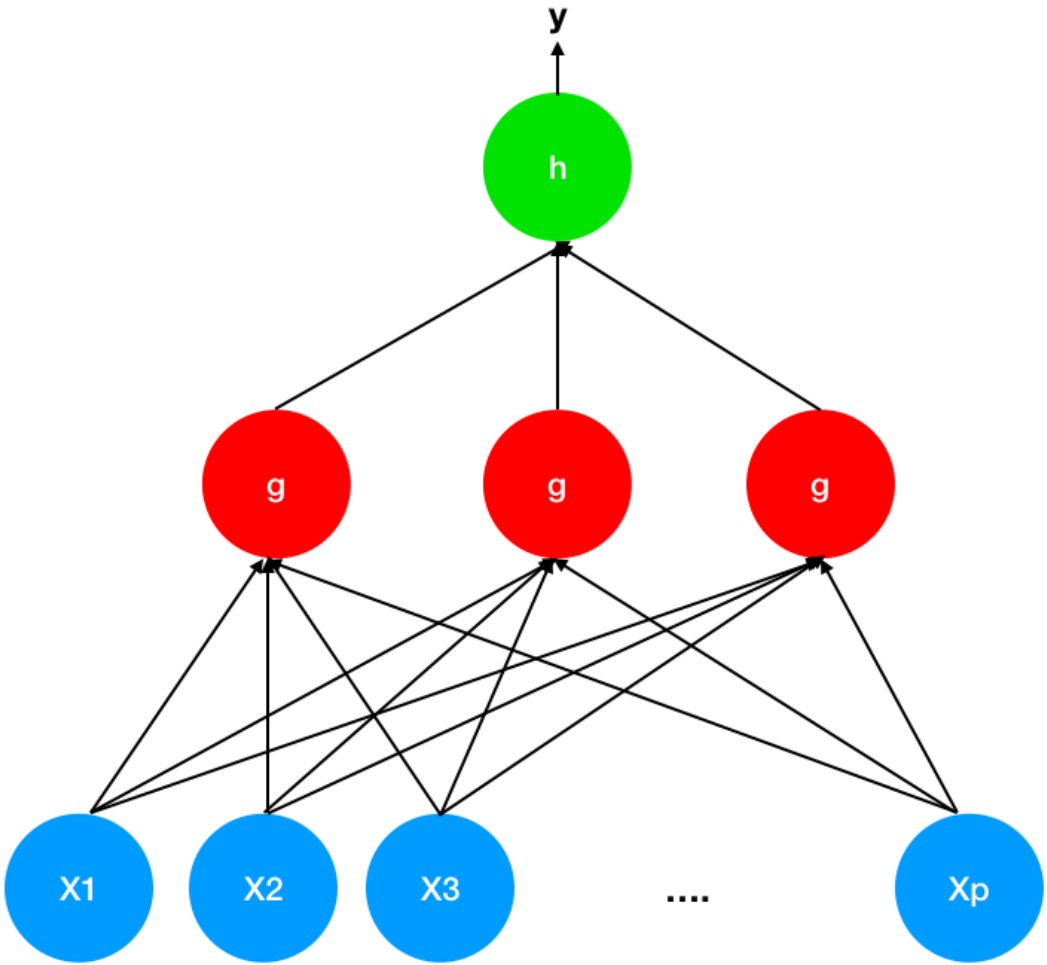
$$y = \phi(x;\theta)^T\beta = \sum_{j=1}^{p'} \underbrace{\phi_j(x;\theta)}_{\text{learned features}} \beta_j$$

We cannot use linear ϕ_j since that just re-creates a linear model (not interesting).
So we instead use:

$$\phi_j(x;\theta) = g(x^T\theta_j + c)$$

where g is some non-linear function (often sigmoid, or ReLU).

Recap: Representing Classifiers/Regressors as Networks



Recap: *Deep* Neural Networks

To get more complex features, we can just recurse! Build more layers of features, each of which is a non-linear function of a linear transformation of the previous.

This is the basic idea of *feed-forward neural networks* or *multilayer perceptrons*.

Recap: Fitting the Model

The basic idea:

- We use a loss function to measure how well we are fitting, and then try to find weights that make our loss small (back-propagation) .

For continuous outputs, we might look $\|y - \widehat{y}\|_2^2$.
For binary or multiple category outputs, we might look at the log likelihood of y_i:

$$\text{Logistic:} \quad \begin{cases} \log\left(\frac{1}{1+e^{-w^T h}}\right) & y_i = 1 \\ \log\left(\frac{e^{-w^T h}}{1+e^{-w^T h}}\right) & y_i = 0 \end{cases}$$

$$\text{Multinomial:} \quad \log\left(\frac{e^{w_{y_i}^T h}}{\sum_j e^{w_j^T h}}\right)$$

Roughly, get the output of the network to match the y we are trying to predict, on the training data.